John von Neumann Institute for Computing


# Unified Scheduling of I/O- and Computation-Jobs for Climate Research Environments

N. Peter Drakenberg, Sven Trautmann

published in

*Parallel Computing: Architectures, Algorithms and Applications* ,
C. Bischof, M. Bücker, P. Gibbon, G.R. Joubert, T. Lippert, B. Mohr, F. Peters (Eds.),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **38**, ISBN 978-3-9810843-4-4, pp. 329-336, 2007.
Reprinted in: Advances in Parallel Computing, Volume **15**,
ISSN 0927-5452, ISBN 978-1-58603-796-3 (IOS Press), 2008.

© 2007 by John von Neumann Institute for Computing
Permission to make digital or hard copies of portions of this work for personal or classroom use is granted provided that the copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise requires prior specific permission by the publisher mentioned above.

http://www.fz-juelich.de/nic-series/volume38

# Unified Scheduling of I/O- and Computation-Jobs for Climate Research Environments

**N. Peter Drakenberg**[1] **and Sven Trautmann**[2]

[1] German Climate Computing Center
Bundesstraße 55, 20146 Hamburg, Germany
*E-mail: drakenberg@dkrz.de*

[2] Max Planck Institute for Meteorology
Bundesstraße 53, 20146 Hamburg, Germany
*E-mail: sven.trautmann@zmaw.de*

Scheduling I/O- and computation jobs is a key factor to efficiently operate large cluster systems. After outlining the necessity of scheduling I/O- and computation jobs in a climate research environment, we present a methodology to schedule I/O jobs depending on the current load on a parallel file system (Lustre). Starting with a system description and some file system benchmarks, we present how to integrate an I/O load sensor into the Sun grid engine. Furthermore we exemplary present how the I/O job scheduling is realized on the *Tornado* Linux cluster system at the German Climate Computing Center (DKRZ) in Hamburg.

## 1 Introduction

Climate simulation is not only used to predict global warming and its consequences, just like weather prediction, climate simulation is of significant importance to many areas of human society. Notable current and future examples being prediction of outbreaks of infectious diseases such as cholera[1] and malaria[2], prediction of agricultural conditions[3], and of course prediction of "difficult" seasonal weather conditions.[4]

In order to achieve reasonable run times (and waiting times), climate simulations are typically run on high-performance computer systems (*e.g.*, vector supercomputers or large clusters), and in addition to using large amounts of processor time, climate simulations also consume and produce large volumes of data.

The cluster system recently installed at DKRZ, provides more than 1000 processor cores, a parallel file system (*Lustre*[5]) with a capacity of about 130 Terabytes, and runs the Linux operating system on all nodes. The cluster has had to be integrated into an existing environment with a shared file system (Sun QFS/SamFS) providing about 75 Terabyte of storage capacity and an underlying storage system with over 4 Petabyte capacity.

As a user front-end to the cluster we use the Sun Grid Engine (SGE)[6]. The SGE is a resource management tool, the purpose of which is to accept *jobs* submitted by users, and schedule the jobs to run on appropriate systems available to it and in accordance with defined resource management policies.

In this paper, we present and explain a sample climate model simulation run. Followed by a description of some system details, we present benchmarks to decide which version of data transfer should be used by I/O jobs. We then describe how data moving jobs can be tightly integrated into the Sun Grid Engine and be scheduled to minimize impact on computation jobs. For user convenience, we provide a set of small tools which enable

the user to copy/move data from one file system to the other. We conclude the paper by discussing future improvements to our I/O job scheduling approach.

## 2 Climate Research Scenarios

As in most sciences, earth-system researchers and meteorologists (*e.g.*, at national weather agencies) use partial differential equations to describe the behaviour of the earth and/or its constituent components (*e.g.*, oceans, atmosphere, *etc*). For example, the evolution of momentum in sea-water (*i.e.*, in oceans) is often expressed as:

$$\frac{\partial}{\partial t}\mathbf{v} + f(\mathbf{k}\times\mathbf{v}) + g\nabla\zeta + \mathbf{v}\cdot\nabla\mathbf{v} + w\cdot\frac{\partial}{\partial z}\mathbf{v} = -\frac{1}{\rho_0}\nabla p + \mathbf{F}_{\mathrm{H}} + \mathbf{F}_{\mathrm{V}},$$

$$\frac{\partial\zeta}{\partial t} + \nabla\left(\int_{z=-H}^{z=\zeta}\mathbf{v}\,\mathrm{d}z\right) = 0,$$

$$\frac{\partial p}{\partial z} = -g\rho$$

where $(\mathbf{v}, w)$ is the velocity vector in a spherical coordinate system $(\lambda, \theta, z)$, $f$ ($f = f(\theta)$) is the Coriolis parameter, $\mathbf{k}$ is an upward vertical unit vector, $g$ is the gravitational acceleration of the earth, $\zeta$ is the sea surface elevation, $\rho_0$ and $\rho$ are the mean sea water density and deviations from that density, respectively, and $p$ is the internal pressure.

Earth-system research and meteorology are primarily concerned with relatively large-scale phenomena. The effects of small-scale phenomena are accounted for through empirically determined parameters that express the ensemble effect of sub-grid scale phenomena in terms of the resolved grid-scale variables. The combination of a set of equations, values of associated parameters, and the method and details of discretization is referred to as a climate/ocean/atmosphere *model*. The execution of a program that numerically solves the discretized equations of some model is referred to as a *model run*.

Model runs are associated with large (for low resolutions) to vast (for high resolutions) volumes of data. To begin with, the (irregular) domains must be specified on which the unknown functions, such as $\mathbf{v}$, $\zeta$, *etc* (*vide supra*), are defined. Additional values are associated with parameters and boundary conditions, and time-varying boundary conditions (*e.g.*, prescribed sea-surface temperatures for an atmosphere model run) in particular. During the model run, the values of unknowns must be maintained for all grid-points/mesh-cells in their domains of definition.

Finally, the values corresponding to the properties of interest are written to files at regular intervals, typically every 10th or every 20th time step. For most model-runs, the generated output is several orders of magnitude larger than all input combined, and illustrates the climate research community's need for high-speed high-capacity file systems.

As a concrete example we consider the model runs performed by the Max-Planck-Institute for Meteorology at the German Climate Computing Center (DKRZ) for the Fourth Assessment Report (AR4)[7] of the Intergovernmental Panel on Climate Change (IPCC). In agreement with the requirements of the IPCC, model runs were performed for three scenarios, denoted A1B, A2 and B1. Using the ECHAM5[8]/MPI-OM[9] coupled model at the T63L31 resolution, 18 individual experiments were realized with a total simulated time of approximately 5000 years. The corresponding model runs consumed 400,000 CPU-hours (on a NEC SX-6[10] system) and produced 400 Terabytes of data. The 192 processor

SX-6 system at DKRZ has a peak performance of 1.5 Teraflop/s and typically runs climate models with $\sim 30\%$ efficiency. The cluster on the other hand, has a peak performance of 5.8 Tflop/s, but typically runs climate models with only $\sim 5\%$ efficiency. Thus, we expect the cluster to be capable of a sustained data-production rate of $\sim 125$ Gigabyte/hour, when performing the type of computations for which it is primarily intended.

## 3 System Description

The system we used for our study is the *Tornado* Linux cluster system operated by the German Climate Computing Center (DKRZ) in Hamburg, Germany. It consists of

- 256 dual core dual socket compute nodes (Sun Fire X2200),
- five dual core eight socket SMP head nodes (Sun Fire X4600) and
- a number of additional nodes providing an management infrastructure and a Lustre file system.

In this paper, we focus on the Lustre file system[5] and two dedicated data mover nodes. A Lustre file system (version 1.6) consists of a number of object storage targets and one meta data target for each of the provided file systems. The object storage targets belonging to one file system are combined to form a logical volume. The *Tornado* cluster provides two Lustre file systems with a combined capacity of approximately 130 TB. The underlying hardware system consists of eight Sun Fire X4500 devices with 48 $\times$ 500 GB hard disks each and a fail over meta data server configuration. The Lustre file systems are accessible from all compute and head nodes using a DDR-InfiniBand interconnect.

Since the Tornado cluster system had to be integrated into an existing infrastructure providing QFS file systems and an underlying HSM system, we had to establish a gateway between the cluster and HSM system. For this purpose the *Tornado* cluster system is equipped with two dedicated data mover nodes (Sun Fire X4200) with access to both the Lustre file system and the QFS (see Fig. 1).

Our first working data mover configuration was running SuSE Linux 9 with service pack 2. The read and write performance to the QFS was impressive with up to 380 Megabyte per second for writing and 325 Megabyte per second for reading operations on an multi-path QFS configuration. To our knowledge, this represented the first installation where *both* Lustre and QFS could be used on one machine with read and write access.

Due to some bugs in the old InfiniBand stack, we had to update the whole cluster system to a newer version, and the new InfiniBand stack was no longer running on SLES 9 SP 2. Since QFS is only supported on a limited number of Linux distributions, we had to update the data mover nodes to SLES 10 and QFS to version 4.6. Unfortunately, we have not yet reached the performance of the previous installation (see next sections for more details). Another drawback of the new QFS version is that we have to use a patchless Lustre client, which has a negative influence on the Lustre file system performance.

## 4 File System Benchmarks

From a performance point of view, the system's hardware gives an upper limit on the amount of data which can be transferred from one file system to the other in a given period

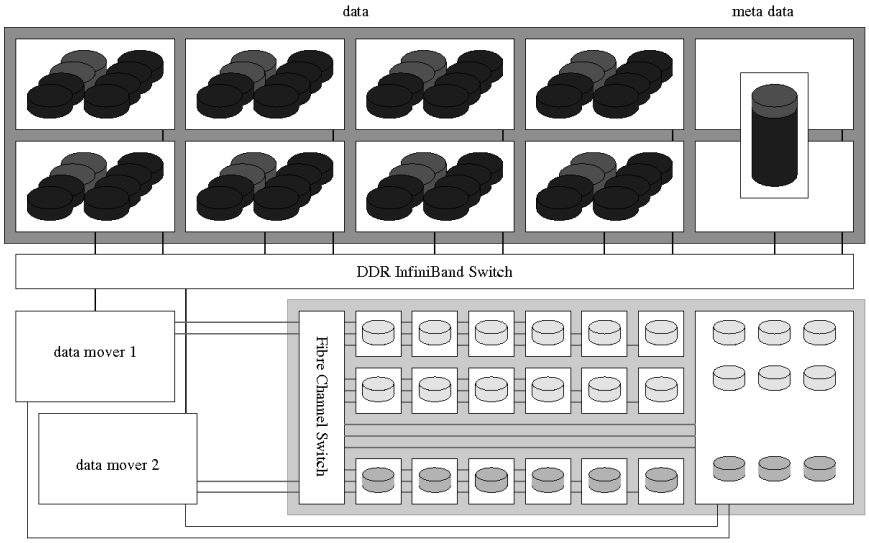


Figure 1. File systems of the *Tornado* Linux cluster system

of time. Both data mover nodes are equipped with two dual-core AMD Opteron processors, one InfiniBand host channel adapter (HCA) and one dual-port Fiber-Channel host bus adapter (HBA) providing two 2 Gigabit ports to the QFS storage network. The InfiniBand HCA limits the throughput to 2 Gigabyte per second. In contrast, the Fiber-Channel HBA allows only 400 Megabyte per second if both ports can be utilized.

To get conclusive performance results we had to study different scenarios. We started by measuring the data rates which can be achieved on each file system. Since the file systems are in production mode already (and remain so), we have always load influences. The second step was to find the best way to copy/move data from one file system to the other, depending on the number of files to transfer and their size. In the third step we studied if and how the concurrent access to both file systems influenced the throughput.

For raw performance measurements we started with the *IOZone*[11] file system benchmark tool which measures the performance for file systems at a time. In Fig. 2 the output of the IOZone benchmark is shown. Lustre delivers up to 1.2 GB/s when reading from the file system and more than 600 MB/s when writing to the file system. In contrast, reading from the QFS is limited to about 180 MB/s. Writing to the QFS can be done with up to 600 MB/s. We assume cache effects here due to hardware limits. The read and write performance of the Lustre file system depends on the file size to transfer[a].

Based on this results we study different methods to transfer data between the file systems by comparing ways of moving data on operating system level with varying parameter sets. As a starting point we use the systems `cp` command and study which transfer rates can be achieved depending on the file size. Table 1 shows the results. In comparison with

[a]File systems details are as follows: *qfs1* file systems provides 44 TB of data with currently 70% capacity utilization. *qfs2* has a size of 32 TB with 94% used. The Lustre file systems provide 32 TB and 94 TB with an utilization of 9% and 32%, respectively.

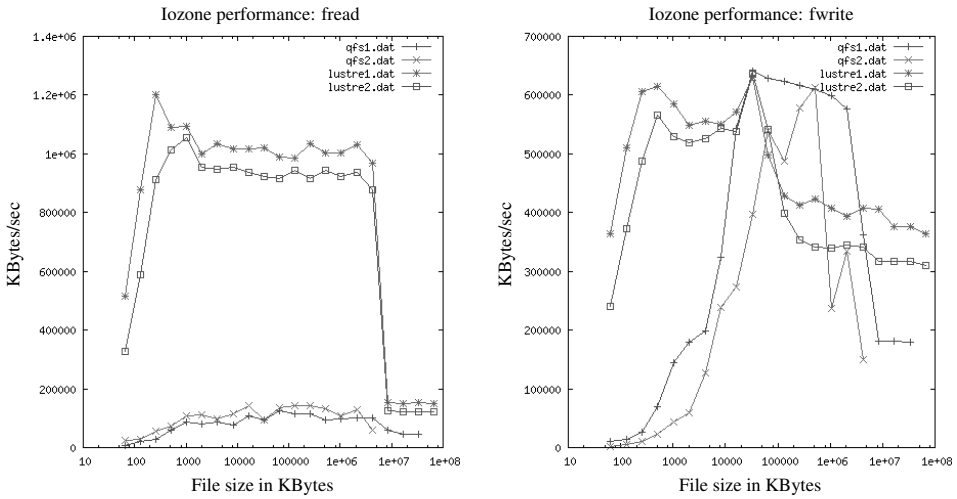


Figure 2. IOZone file system benchmark for QFS and Lustre file systems on *Tornado*

| file systems used | 1M | 4M | 16M | 64M | 256M | 1G | 4G | 16G | 64G |
|---|---|---|---|---|---|---|---|---|---|
| qfs1 → lustre1 | 21.7 | 30.8 | 37.6 | 38.3 | 56.7 | 79.4 | 74.7 | 65.3 | 66.5 |
| qfs1 → lustre2 | 50.0 | 61.5 | 80.0 | 93.0 | 102.0 | 79.0 | 79.1 | 74.5 | 68.1 |
| qfs2 → lustre1 | 22.2 | 30.1 | 38.3 | 35.1 | 32.4 | 35.0 | 31.5 | 30.0 | 30.6 |
| qfs2 → lustre2 | 20.4 | 20.5 | 30.0 | 30.6 | 36.0 | 29.6 | 29.6 | 31.7 | 29.7 |
| lustre1 → qfs1 | 9.0 | 28.8 | 58.4 | 76.7 | 81.8 | 80.2 | 81.9 | 87.3 | 93.2 |
| lustre1 → qfs2 | 5.9 | 20.6 | 36.8 | 51.4 | 61.0 | 73.8 | 73.2 | 80.4 | 76.5 |
| lustre2 → qfs1 | 11.4 | 25.8 | 50.6 | 69.3 | 76.5 | 76.9 | 79.0 | 79.6 | 79.6 |
| lustre2 → qfs2 | 8.5 | 18.3 | 34.9 | 55.0 | 65.9 | 65.3 | 63.2 | 69.4 | 66.0 |

Table 1. Throughput using cp in MB/s

the IOZone benchmark the results achieved using the `cp` command are much lower. To get better performance, we study if and how using a different block sizes can improve the throughput. For this purpose we use the `dd` command which allows to specify different block size for input and output.

We studied all eight file system combinations as before for the `cp` command, five different file sizes (16M, 256M, 1G, 4G, 16G) and six different block sizes (4K, 16K, 64K, 256K, 1M, 4M) for both input and output. Table 2 shows the maximum throughput and Table 3 the corresponding values for input and output block size.

The maximum throughput was reached when copying data from the Lustre file systems to the *qfs1* file system. Since the throughput is much higher for the `dd` command we decided to use it instead of the `cp` command. Since the main purpose of the data mover nodes is to copy data from the Lustre file system to the QFS, we decided to use a block size of 4 Megabyte for reading the Lustre file system and a block size of 256k to write to the QFS.

| file systems used | 16M | 256M | 1G | 4G | 16G |
|---|---|---|---|---|---|
| qfs1 → lustre1 | 116.0 | 114.0 | 95.8 | 88.8 | 83.8 |
| qfs1 → lustre2 | 119.0 | 113.0 | 112.0 | 91.5 | 87.0 |
| qfs2 → lustre1 | 34.3 | 113.0 | 32.9 | 35.5 | 37.6 |
| qfs2 → lustre2 | 120.0 | 116.0 | 103.0 | 91.9 | 36.9 |
| lustre1 → qfs1 | 60.8 | 173.0 | 85.3 | 111.0 | 83.2 |
| lustre1 → qfs2 | 47.4 | 111.0 | 68.2 | 87.5 | 74.1 |
| lustre2 → qfs1 | 81.3 | 184.0 | 108.0 | 123.0 | 104.0 |
| lustre2 → qfs2 | 51.0 | 117.0 | 85.9 | 95.7 | 89.5 |

Table 2. Maximum throughput (in MB/s) reached using the `dd` command

| file systems used | 16M | 256M | 1G | 4G | 16G |
|---|---|---|---|---|---|
| qfs1 → lustre1 | 64k/256k | 16k/256k | 64k/256k | 16k/256k | 1m/256k |
| qfs1 → lustre2 | 64k/64k | 4k/256k | 64k/256k | 64k/64k | 256k/256k |
| qfs2 → lustre1 | 256k/256k | 256k/256k | 64k/1m | 256k/64k | 64k/256k |
| qfs2 → lustre2 | 256k/256k | 256k/64k | 64k/64k | 256k/256k | 256k/256k |
| lustre1 → qfs1 | 256k/256k | 4m/256k | 256k/16k | 256k/64k | 1m/256k |
| lustre1 → qfs2 | 4m/256k | 4m/64k | 4m/1m | 4m/64k | 1m/64k |
| lustre2 → qfs1 | 256k/64k | 64k/64k | 256k/4k | 256k/16k | 256k/64k |
| lustre2 → qfs2 | 256k/64k | 4m/16k | 4m/64k | 1m/256k | 1m/64k |

Table 3. Input/output buffer size used to reach maximum throughput using the `dd` command

## 5 Grid Engine Integration

In its standard configuration, the SGE keeps track of and manages resources such as CPU time, physical memory, process segment sizes, load averages, *etc*. The scheduling of a job is determined by factors such as the job's resource requirements, the job's waiting time, the job's deadline (if any), the job owner's assigned share entitlements of resources, *etc*.

In our configuration, the SGE also keeps track of the I/O load on the Lustre file system and uses this information to run data transfer jobs on the combined Lustre/QFS nodes (see above) at times when the I/O load on the Lustre file system (due to compute jobs) is not high. The Lustre I/O load was made visible to the SGE by defining a so-called consumable attribute, named `ioload`, in the SGE (see [12, pp. 67–86]) and by implementing a custom load sensor. Every 30 seconds, the custom load sensor gathers the volume (no. of bytes) of data read and written per second on each object storage server (OSS) of the Lustre file system (this information is made available by Lustre in `/proc/fs/lustre/ost/OSS/ost_io/stats` on each OSS). With the volume of data read and written on the $i$th OSS ($i \in \{1,...,8\}$) denoted by $dr_i$ and $dw_i$, respectively, the load sensor calculates the currently available Lustre I/O capacity $ioc$ as:

$$ioc = 1000.0 \; - \; \{\max_{1 \le i \le 8}(w_{\mathrm{r}} dr_i + w_{\mathrm{w}} dw_i)\}/2^{20} \qquad (5.1)$$

and reports this value to the SGE as the current value of the attribute `ioload`. In Eq. (5.1), $w_{\mathrm{r}}$ and $w_{\mathrm{w}}$ are weight factors (at present $w_{\mathrm{r}} = 1.0$, $w_{\mathrm{w}} = 2.0$), and the value $1000.0$ was

chosen as the available I/O capacity of an idle Lustre file system based on the file system I/O benchmark results reported in Section 4.

In addition to the matters described above, an SGE job queue named `qfs` has been defined, for which the combined Lustre/QFS nodes are the only execution hosts. The `qfs` queue is configured with `ioload` as a consumable attribute (base value: 1000.0), and with `ioload` values of 200.0 and 100.0 as load- and suspend-thresholds, respectively. As a result of this setup, the specified use of the `ioload` resource by jobs running in the `qfs` queue is deducted from the current value of the `ioload` consumable (while each job is running), and the specified thresholds prevent the `qfs` queue from overusing the Lustre file system. Furthermore, and more importantly, the load- and suspend-thresholds are applied to the minimum of the `ioload` sensor value and the `ioload` consumable, and will thereby prevent jobs in the `qfs` queue form starting or suspend running data transfer jobs, in order to keep the available Lustre I/O capacity at a sufficient level to minimize the impact on running computations.

## 6 Experiences

Model runs at DKRZ are typically performed as job-chains, with the last action of a running job being to submit a new instance of itself that resumes the model run where the current job left off, occasionally interleaved with (non-parallel) post-processing jobs and/or data transfer operations. For this context and environment, the commands `qfs-cp` and `qfs-mv` have been implemented to transfer data between the Lustre and QFS file systems. These commands first determine the volume of data and the expected duration of the transfer, and then create a job-script to perform the actual file transfers, that is subsequently submitted to the SGE with the consumption of the `ioload` resource and an upper limit on run-time specified. A benefit of this scheme is that it enables data transfers to be decoupled from computations, allowing both to proceed independently and in parallel (also for data transfers initiated by computational jobs).

## 7 Conclusion and Future Work

As mentioned in Section 5, the purpose of the work described here has been to find a viable way of transferring data away from the cluster's file-systems, while minimizing the impact on computations. In other words, our objective has been to schedule and run pure I/O jobs without disturbing compute jobs.

The primary constraint in the present context is a lack of information.[b] Ideally (given the repetitive nature of our computations), each compute job would indicate its patterns of interconnect usage (*e.g.*, MPI communication) as well as how often and how much output it will generate. The fact that information of this kind will not be provided by queued and/or running jobs, has led us to adopt the relatively simple scheme for I/O-job

[b] For pure compute jobs, detailed and accurate information about expected resource utilization, appears not to be essential for good job scheduling.[13] In part, this is due to the implicitly uniform use of the primary resource (*i.e.*, CPUs). I/O (by compute jobs), on the other hand, is often done in bursts at regular intervals that are well separated in time (*e.g.*, after every $n$ timesteps, for some fixed $n$), and accurate information about such behavioural properties (of compute jobs) would clearly be useful for scheduling I/O jobs.

scheduling described above. However, despite the simplicity of our scheme, we have found the SGE to be insufficiently flexible for some of our needs, and these limitations appear to be present also in many other job-scheduling systems. In particular, persistent increases *and* decreases of resources (*e.g.*, available disk space) as consequences of scheduling and running jobs can not be properly handled by the SGE.

In our work thus far, we have introduced a Lustre I/O load sensor to the *Sun Grid engine*, developed user-level commands that handle I/O job creation and submission, and we have evaluated different means of performing the actual data transfers. As a result of the latter evaluation, the I/O jobs created by our user-level commands use the `dd` command, with optimized parameters, to perform data transfers. For the future we plan to not only monitor the file system load, but also the InfiniBand network load, to decide if an I/O job should be executed at a given point in time or not. This approach will further reduce the effect of I/O jobs on computational jobs, since the same InfiniBand network is used for both I/O and message-passing communication.

## References


1. K. Koelle et al., *Refractory periods and climate forcing in cholera dynamics*, Nature, **436**, 696–700, (2005).
2. M. C. Thomson et al., *Malaria early warnings based on seasonal climate forecasts from multi-model ensembles*, Nature, **439**, 576–579, (2006).
3. F. S. Royce et al., *Model-based optimization of crop management for climate forecast applications*, Trans. Am. Soc. Agr. Eng., **44**, 1319–1327, (2001).
4. D. R. Easterling et al., *Climate extremes: observations, modelling, and impacts*, Science, **289**, 2068–2074, (2000).
5. Cluster File Systems, Lustre homepage, http://clusterfs.com.
6. Sun Microsystems, Gridengine homepage, http://gridengine.sunsource.net/.
7. S. Solomon, D. Qin, M. Manning, M. Marquis, K. B. Averyt, M. Tignor, H. L. Miller and Z. Chen, (Eds.), *Climate Change 2007: The Physical Science Basis.*, (Cambridge University Press, Cambridge, 2007).
8. E. Roeckner et al., *The atmospheric general circulation model ECHAM5: Part I, model description*, Report No. 349, Max-Planck-Institut für Meteorologie, Bundesstraße 53, Hamburg, Germany, (2003).
9. S. J. Marsland, H. Haak, J. H. Jungclaus, et al., *The Max-Planck-Institute global ocean/sea ice model with orthogonal curvilinear coordinates*, Ocean Modelling, **5**, 91–127, (2003).
10. K. Kitagawa, S. Tagaya, Y. Hagihara and Y. Kanoh, *A Hardware Overview of SX-6 and SX-7 Supercomputer*, NEC Res. & Develop., **44**, 2–7, (2003).
11. W. Norcott and D. Capps, IOZone Filesystem Benchmark, http://www.iozone.org.
12. Sun Microsystems, Inc., Santa Clara, CA 95054, *N1 Grid Engine 6 Administration Guide*, (2005), Part No: 817-5677-20.
13. A. W. Mu'alem and D. G. Feitelson, *Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling*, IEEE Trans. Parall. Distrib. Sys., **12**, 529–543, (2001).